1. Figure 1.4 of your text shows the basics while Figure 2.21 shows abit more detail. This HW is over chapter 1 so let's talk about the basics.

The compiler translates the high-level programming language such as C/C++ into an assembly language program specific to the target ISA, such as MIPS.

The assembler takes the assembly languange program and converts it into machine language which is a binary representation of the assembly using the instruction formats described in the ISA. More advanced discussion:
The ~~[illegible]~~ linker patches together seperately assembled object files into one executable and the loader loads the executable into memory so the processor can begin execution.

2. a) die size is 18.9mm * 13.6mm
wafer diameter is 300mm

The following equation was provided in the lecture slides:

$$\text{Dies per wafer} = \frac{\pi * (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi * \text{wafer diameter}}{\sqrt{2 * \text{Die area}}}$$

$$= \frac{\pi * (300mm/2)^2}{18.9mm * 13.6mm} - \frac{\pi * 300mm}{\sqrt{2 * 18.9mm * 13.6mm}}$$

$$= 275.00 - 41.568 = 233.43 \text{ dies}$$

so 233 dies

2 b)

$$\text{die yield} = \text{wafer yield} \times \frac{1}{\left(1 + \frac{\text{defects}}{\text{area}} \times \text{die area}\right)^N}$$

$$= 0.99 \times \frac{1}{\left(1 + \frac{0.020 \text{ defects}}{cm^2} \frac{1 cm^2}{100 mm^2} \times 18.9mm \times 13.6mm\right)^7}$$

$$= 0.697 \approx 69.7\%$$

c)

$$\text{cost per die} = \frac{\text{cost per wafer}}{\text{dies per wafer} * \text{die yield}}$$

$$= \frac{\$6{,}500}{233 \text{ dies} * 69.7\%} = \underline{\$40.02} \text{ per die}$$

d) This problem is really just

$$\text{profit} = \underset{\text{price}}{1.5x} - \underset{\text{cost}}{1.0x} = 0.5x \$6{,}500 = \underline{\$3{,}250 \text{ per wafer}}$$

e)

$$\$40.02 \times 1.5 = \$1{,}652.80$$

$$\underset{\text{failed dies}}{233 \tfrac{\text{dies}}{\text{per wafer}} (1 - 0.697)} \underset{\text{\% of failed dies repurposed}}{(0.65)} \underset{\text{\% of full price repurposed dies can be sold for}}{(0.6)} (\text{full price per die}) =$$

total gains per wafer $= \$3{,}250.00 + 1{,}652.80 = \$4{,}902.80$

left over after R&D $= 0.85 * \$4902.80 = \underline{\$4{,}169.40}$

3

a) $\underline{P_3}$ has the highest clock rate @ 3.2 GHz

clock period of each:

P_1 clock period $= \frac{1}{1.8\,GHz} = 0.555ns = 555ps$

P_2 clock period $= \frac{1}{2.4\,GHz} = 0.4167\,ns =$ ~~[illegible]~~ $416.7ps$

P_3 clock period $= \frac{1}{3.2\,GHz} = 0.3125ns = 312.5ps$

b)

$MIPS = \frac{\text{clock rate}}{CPI} \cdot \frac{1}{10^6}$

$MIPS_{P_1} = \frac{1.8\,GHz}{0.8\,cc/instr} \cdot 10^{-6} = 2{,}250\,MIPS$

$MIPS\,P_2 = \frac{2.4\,GHz}{1.2\,cc/instr} \cdot 10^{-6} = 2{,}000\,MIPS$

$MIPS\,P_3 = \frac{3.2\,GHz}{1.6\,cc/instr.} \cdot 10^{-6} = 2{,}000\,MIPS$

P_1 <u>has the highest</u> MIPS

c) $t_{execution} = \frac{\text{dynamic instr. count}}{\text{instr/sec}}$

$t_{ex\,P_1} = \frac{22.0\text{ million instr}}{2{,}250\,MIPS} = \underline{9.78\,ms}$

$t_{ex\,P_2} = \frac{19\text{ million instr}}{2{,}000\,MIPS} = \underline{9.5\,ms}$ <u>P_2 is faster!</u>

$t_{ex\,P_3} = \frac{22.0\text{ million instr.}}{2{,}000\,MIPS} = \underline{11.0\,ms}$

3 d) the reference machine takes 35ms

$$\text{speedup}_{P_1} = \frac{35ms}{9.78ms} = \underline{3.58x}$$

$$\text{speedup}_{P_2} = \frac{35ms}{9.5ms} = \underline{3.68x}$$

$$\text{speedup}_{P_3} = \frac{35ms}{11.0ms} = \underline{3.18x}$$

4. a) This system is I/O bound because it spends more than 50% of its time waiting on I/O

b) $$t_{improved} = \frac{t_{affected}}{\text{Speedup}_{affected}} + t_{unaffected}$$

let's assume the program originally took a second

$$t_{improved} = \frac{(0.4)(1s)}{20} + (0.6)(1s)$$

$$= 0.62s$$

$$\text{speedup} = \frac{1s}{0.62s} = \underline{1.61x}$$

c)

$$t_{improved} = \frac{0.6s}{3} + 0.4s$$

$$= 0.60s$$

$$\text{speedup} = \frac{1s}{0.6s} = \underline{1.67x}$$

d) You can speedup disk access by using a parallel disk array, using a disk with a higher RPM, or by replacing disk with flash.

5. a) $t_{new} = \frac{800s}{4} + 200s = \underline{400s}$

$\text{speedup} = \frac{1000s}{400s} = \underline{2.5x}$

b) $t_{new} = \frac{800s}{8} + 200s = \underline{300s}$

$\text{speedup} = \frac{1000s}{300s} = \underline{3.33x}$

c) $t_{new} = \frac{800s}{\infty} + 200s = \underline{200s}$

$\text{speedup} = \frac{1000s}{200s} = \underline{5x}$

d) $t_{new} = \frac{900s}{\infty} + 100s = \underline{100s}$

$\text{speedup} = \frac{1000s}{100s} = \underline{10x}$

6.

a) $t_{new} = \frac{2s(0.95)}{\infty} + 2s(1-0.95)$

$= 0 + 0.1s$

$\text{speedup} = \frac{2s}{0.1s} = \underline{20x}$

b) $t_{new} = \frac{2s(0.95)}{50} + 2s(1-0.95)$

$= \underline{0.138s}$

$\text{speedup} = \frac{2s}{0.138s} = \underline{14.493x}$

6. c) Energy = power × time

the time will be reduced by 14.493x
while the power will be doubled

$$\frac{\text{Energy old}}{\text{Energy new}} = \frac{14.493}{2} = \underline{7.25\times}$$

d) See attached graph and GNU plot source file for generating the graph

7. a) $P_{total} = P_{dynamic} + P_{static}$
$= 1.2W$

Energy per operation $= P_{total} \times t_{operation}$
$= 1.2W \times 1s/_{op} = 1.2J/op$

b) In this case dynamic power will change as well as the operation time

$P_{dynamic} \propto cV^2f$ thus $P_{new\ dynamic} = 1.0W \cdot \frac{250MHz}{500MHz}$
$= 0.5W$

Note that static power does ~~not~~ dependend on f and thus does not change

$t_{new} = 1s \cdot \frac{500MHz}{250MHz} = 2s$

$\underline{P_{total} = 0.5W + 0.2W = 0.7W}$

Energy/op = $(0.5W + 0.2W)(2s) = \underline{1.4 J/s}$

notice the energy per op went up due to longer execution time burning more static energy!

7. c) In this case, we also reduce the voltage from 1.2V to 0.9V, taking advantage of the V^2 relationship to dynamic power

$$P_{dynamic} = 1.0W \times \frac{250MHz}{500MHz} \times \left(\frac{0.9V}{1.2V}\right)^2 = 0.281W$$

$$P_{static} = V \cdot I_{leakage} = 0.2W \frac{0.9V}{1.2V} = 0.15W$$

$$P_{total} = 0.281W + 0.15W = \underline{0.431W}$$

$$\text{Energy per op} = 0.431W \times 2s = \underline{0.863 J/op}$$

Now we see an energy savings! but only when we drop voltage as well as frequency!

d) This problem is the ~~two~~ reverse of the previous.

$$P_{dynamic} = 1.0W \times \frac{650MHz}{500MHz} \times \left(\frac{1.4V}{1.2V}\right)^2 = 1.7694W$$

$$P_{static} = 0.2W \times \frac{1.4V}{1.2V} = 0.2333W$$

$$t_{new} = 1s \frac{500MHz}{650MHz} = 0.769s$$

$$P_{total} = 1.7694W + 0.2333W = \underline{2.003}W$$

$$\text{Energy per op} = 2.00W \times 0.769s = \underline{1.54 J/op}$$

e) Power gating is one technique that removes power from unused logic. For example, an unused floating point unit or even an entire core might be powered down. This saves dynamic and static energy!

Clock gating disables the clock to unused logic. This only saves dynamic energy but keeps static elements powered

7. e) continued..

such as memory and registers. This allows a given logic block to retain state while saving power.

Clock gating does not save as much energy as power gating but allows logic to come alive more quickly.

Other techniques include ~~hardware~~ hardware acceleration where commonly used operations are mapped into more energy efficient hardware.

A modern system on a chip (SoC) will use <u>all</u> techniques discussed to save energy!

f) The previous problems changed V and f but left C alone. This problem changes all three!

$$P_{dynamic} = 1.0W \left(\frac{130nm}{90nm}\right)\left(\frac{1.5V}{1.2V}\right)^2\left(\frac{333\,MHz}{500\,MHz}\right) = \underline{1.5W}$$

capacitance increase   voltage increase   frequency goes down!

$$P_{static} = \left(\frac{1.5V}{1.2V}\right) 0.2W = 0.25W$$

$$P_{total} = 1.5W + 0.25W = \underline{1.75W}$$

$t_{new} =$ ← time increases!

$$= 1s \times \frac{500MHz}{333\,MHz} = 1.5015s$$

Energy per op =

$$1.75W \times 1.5015s = \underline{2.63\ J/op}$$

$$\frac{2.63J/op}{1.2\ J/op} = \underline{2.19x} \text{ higher!}$$

8. **(10 points)** The execution times of a few SPEC2006Cint benchmarks on three different machines are shown in the table below.

Benchmark	Ultra 5 Time (sec)	Machine A Time (sec)	Machine A SPEC ratio	Machine B Time (sec)	Machine B SPEC ratio
perlbench	9,770	454	21.52	253	38.62
bzip2	9,650	520	18.56	438	22.03
gcc	8,050	461	17.46	234	34.40
mcf	9,120	268	34.03	150	60.80
gobmk	10,490	429	24.45	383	27.39
hmmer	9,330	197	47.36	135	69.11
sjeng	12,100	529	22.87	362	33.43
libquantum	20,720	91	227.69	7	2960.00
h264ref	22,130	599	36.95	427	51.83
omnetpp	6,250	305	20.49	163	38.34
astar	7,020	327	21.47	234	30.00
xalancbmk	6,900	253	27.27	117	58.97
Geometric mean			30.45		56.98

(a) Using the Ultra 5 as a baseline, fill in the SPECratio columns above for Machines A and B.

(b) Compute the geometric mean of the SPECratios for both machines. see table above

(c) Given the benchmarks above, which machine (A or B) is faster? By how much?

(d) libquantum has fallen victim to substantial compiler optimizations as evidenced by such extraordinary speedups over the reference machine. Consequently, it has been removed from SPEC2017. Recalculate the geometric mean of the normalized performance without libquantum. Compare machine A to machine B excluding libquantum.

c) Machine B is faster because it has the higher spec ratio. Specifically Machine B is faster than Machine A by

$$\frac{56.98}{30.45} = 1.87x$$

d)

Machine A
Geometric mean: 25.36

Machine B
Geometric mean: 39.79

Machine ~~A~~ B is still faster but by a lesser amount:

$$\frac{39.79}{25.36} = 1.57x$$

9. a) CPI = clock cycles per instr

Program A instr count = $8 \cdot 10^8$ instr
execution time = 1.1 s

Program B instr count = $1.2 \cdot 10^9$ instr
execution time = 1.6 s

both run on processor w/ $\frac{1}{1ns}$ = 1 GHz clock rate

using units to derive equation:

$$CPI = cc/instr = \frac{cc}{s} \times s \times \frac{1}{instr}$$

$$CPI_A = 10^9\, cc/s \times 1.1\, s \times \frac{1}{8 \cdot 10^8\, instr} = \frac{1.1}{8} \cdot 10\ cc/instr$$

$$= \underline{1.375\ cc/instr}$$

$$CPI_B = 10^9\, cc/s \times 1.6\, s \times \frac{1}{1.2 \cdot 10^9\, instr} = \frac{1.6}{1.2}\ cc/instr$$

$$= \underline{1.333}\ cc/instr.$$

b)

$$t_{execute} = \frac{CPI \times instr.\ count}{clock\ rate}$$

or

$$clock\ rate = \frac{CPI \times instr\ count}{t_{execute}}$$

$$\frac{clock\ rate_A}{clock\ rate_B} = \frac{CPI_A \times instr\ count_A}{t_{execute_A}} \cdot \frac{t_{execute_B}}{CPI_B \times instr.\ count_B}$$

$t_{ex_A} = t_{ex_B}$ so...

$$= \frac{CPI_A \times instr\ count_A}{CPI_B \times instr\ count_B}$$

9. b) continued...

$$\frac{\text{clock rate}_A}{\text{clock rate } B} = \frac{1.375\ \text{cc/instr} \times 8\cdot 10^8\ \text{instr}}{1.333\ \text{cc/instr} \times 1.2\cdot 10^9\ \text{instr}}$$

$$= \underline{0.688}$$

c) original processor ran a 1 GHz

$$t_{ex_C} = \frac{CPI_C \times \text{instr. count}_C}{\text{clock rate}}$$

$$\text{speedup}_{CA} = \frac{t_{ex\,A}}{t_{ex\,C}} = \frac{CPI_A \times \text{instr. count}_A}{\text{clock rate}} \times \frac{\text{clock rate}}{CPI_C \times \text{instr. count}_C}$$

$$= \frac{1.375\ \text{cc/instr} \times 8\cdot 10^8\ \text{instr}}{1.3\ \text{cc/instr} \cdot 7.5\cdot 10^8\ \text{instr}} = \underline{1.13x}$$

$$\text{speedup}_{CB} = \frac{1.333\ \text{cc/instr} \cdot 1.2\cdot 10^9\ \text{instr}}{1.3\ \text{cc/instr} \cdot 7.5\cdot 10^8\ \text{instr}} = \underline{1.64x}$$